

日 本 国 特 許 庁
JAPAN PATENT OFFICE

別紙添付の書類に記載されている事項は下記の出願書類に記載されている事項と同一であることを証明する。

This is to certify that the annexed is a true copy of the following application as filed with this Office.

出 願 年 月 日 2 0 0 4 年 3 月 3 日
Date of Application:

出 願 番 号 特 願 2 0 0 4 - 0 5 9 8 6 5
Application Number:
[ST. 10/C]: [J P 2 0 0 4 - 0 5 9 8 6 5]

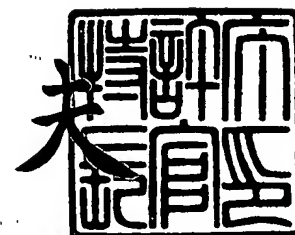
出 願 人 オムロン株式会社
Applicant(s):

CERTIFIED COPY OF
PRIORITY DOCUMENT

2 0 0 4 年 3 月 2 4 日

特許庁長官
Commissioner,
Japan Patent Office

今 井 康 夫



【書類名】 特許願
【整理番号】 62947
【提出日】 平成16年 3月 3日
【あて先】 特許庁長官殿
【国際特許分類】 G05B 19/05
【発明者】
 【住所又は居所】 京都府京都市下京区塩小路通堀川東入南不動堂町 8 0 1 番地 オムロン株式会社内
 【氏名】 岡 実
【発明者】
 【住所又は居所】 京都府京都市下京区塩小路通堀川東入南不動堂町 8 0 1 番地 オムロン株式会社内
 【氏名】 出来 仁太郎
【発明者】
 【住所又は居所】 京都府京都市下京区西洞院木津屋橋通東入ル オムロンソフトウェア株式会社内
 【氏名】 矢尾板 宏心
【発明者】
 【住所又は居所】 京都府京都市下京区塩小路通堀川東入南不動堂町 8 0 1 番地 オムロン株式会社内
 【氏名】 市村 勝彦
【発明者】
 【住所又は居所】 京都府京都市下京区塩小路通堀川東入南不動堂町 8 0 1 番地 オムロン株式会社内
 【氏名】 小野 彰男
【特許出願人】
 【識別番号】 000002945
 【氏名又は名称】 オムロン株式会社
 【代表者】 作田 久男
【代理人】
 【識別番号】 100092598
 【弁理士】
 【氏名又は名称】 松井 伸一
【先の出願に基づく優先権主張】
 【出願番号】 特願2003- 68920
 【出願日】 平成15年 3月13日
【手数料の表示】
 【予納台帳番号】 019068
 【納付金額】 21,000円
【提出物件の目録】
 【物件名】 特許請求の範囲 1
 【物件名】 明細書 1
 【物件名】 図面 1
 【物件名】 要約書 1
 【包括委任状番号】 9800459

【書類名】 特許請求の範囲**【請求項 1】**

I/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するCPUユニットを含むプログラマブルコントローラと、そのプログラマブルコントローラに接続され、前記ユーザプログラムを編集するためのプログラミングツールと、かかる制御装置であって、

前記プログラミングツールは、

プログラマブルコントローラに対してユーザプログラムをアップロードおよびダウンロードする手段を持ち、

前記プログラマブルコントローラのCPUユニットは、

ユーザプログラム格納用の2つのメモリと、

サイクリック処理動作中に前記プログラミングツールからユーザプログラム編集処理がなされる場合に、ユーザプログラム実行処理の実行対象をいずれか一方のメモリに格納されているユーザプログラムと定める手段と、

前記プログラミングツールのアップロードによって、サイクリック処理動作の周辺処理期間中に、プログラミングツールに対して一方または他方のメモリのいずれかに格納されているユーザプログラムを出力する手段と、

前記プログラミングツールからダウンロードされてきたユーザプログラムを、サイクリック処理動作の周辺処理期間中に、ユーザプログラム実行処理の実行対象としていない前記他方のメモリへ格納する手段と、

その格納が完了したことを条件に、ユーザプログラム実行処理の実行対象を前記一方のメモリから前記他方のメモリに切り替える手段と、

前記他方のメモリに格納されたユーザプログラムを前記一方のメモリへコピー格納する手段と、

を持つことを特徴とする制御装置。

【請求項 2】

請求項 1 記載の制御装置において、

サイクリック処理動作中に前記プログラミングツールからユーザプログラム編集処理がなされる場合に、ユーザプログラム実行処理の実行対象を、他方のメモリに格納されているユーザプログラムから一方のメモリに格納されているユーザプログラムへ切り替える手段と、

をさらに持つことを特徴とする制御装置。

【請求項 3】

プログラミングツールとの間でユーザプログラムをアップロードおよびダウンロードでき、I/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するCPUユニットであって、

ユーザプログラム格納用の2つのメモリと、

サイクリック処理動作中に前記プログラミングツールからユーザプログラム編集処理がなされる場合に、ユーザプログラム実行処理の実行対象をいずれか一方のメモリに格納されているユーザプログラムと定める手段と、

前記プログラミングツールのアップロードにより、サイクリック処理動作の周辺処理期間中に、前記プログラミングツールに対して一方または他方のメモリのいずれかに格納されているユーザプログラムを出力する手段と、

前記プログラミングツールからダウンロードされてきたユーザプログラムを、サイクリック処理動作の周辺処理期間中に、ユーザプログラム実行処理の実行対象としていない前記他方のメモリへ格納する手段と、

その格納が完了したことを条件に、ユーザプログラム実行処理の実行対象を前記一方のメモリから前記他方のメモリに切り替える手段と、

前記他方のメモリに格納されたユーザプログラムを前記一方のメモリへコピー格納する手段と、

を持つことを特徴とするCPUユニット。

【請求項4】

ユーザプログラムを格納する2つのメモリを持ってI/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するCPUユニットが接続されたプログラマブルコントローラに対して、プログラミングツールから前記ユーザプログラムを編集する方法であって、

CPUユニットにて予め、前記2つのメモリに格納しているユーザプログラムを同一の内容としておき、

CPUユニットにて、プログラミングツールからユーザプログラムを編集処理するのに先立ち、ユーザプログラムの演算実行対象を、いずれか一方のメモリに格納されているユーザプログラムと定めておき、CPUユニットのサイクリック処理動作をそのまま継続させ

、プログラミングツールにて、CPUユニットの動作中に、CPUユニットの一方または他方のメモリのいずれかに格納されているユーザプログラムをアップロードし、そのユーザプログラムを編集し、

プログラミングツールにて、その編集を終了した後、CPUユニットが動作中に、前記プログラミングツールから前記他方のメモリへ編集後のユーザプログラムをダウンロードして格納し、

CPUユニットにて、その格納を終了した後、前記ユーザプログラムの演算実行対象を前記一方のメモリから前記他方のメモリに切り替えて、CPUユニットが編集後のユーザプログラムを実行処理するようにし、

CPUユニットにて、前記他方のメモリに格納された編集後のユーザプログラムを前記一方のメモリに格納して、両メモリのユーザプログラムを同一内容とするようにしたことを特徴とするプログラマブルコントローラのユーザプログラム編集方法。

【請求項5】

請求項4記載のプログラマブルコントローラのユーザプログラム編集方法であって、前記プログラミングツールにおいてユーザプログラムをアップロードおよびダウンロードする際には、その対象メモリを、CPUユニットの動作中におけるユーザプログラム演算実行対象のほうのメモリでないもう一方のメモリとする

ことを特徴とするプログラマブルコントローラのユーザプログラム編集方法。

【請求項6】

請求項4記載のプログラマブルコントローラのユーザプログラム編集方法であって、前記プログラミングツールにおいてユーザプログラムをアップロードおよびダウンロードする際には、CPUユニットの動作中における周辺処理期間中に行うことを特徴とするプログラマブルコントローラのユーザプログラム編集方法。

【請求項7】

請求項4に記載のプログラマブルコントローラのユーザプログラム編集方法であって、新しく次回、プログラミングツールにてユーザプログラムを編集する際、

CPUユニットにて、ユーザプログラムの演算実行対象を前記他方のメモリから前記一方のメモリに切り替え、その状態を編集後のユーザプログラムのダウンロードが終了するまで維持する

ことを特徴とするプログラマブルコントローラのユーザプログラム編集方法。

【請求項8】

請求項4に記載のプログラマブルコントローラのユーザプログラム編集方法であって、新しく次回、プログラミングツールにてユーザプログラムを編集する際、

CPUユニットにて、ユーザプログラムの演算実行対象を前記他方のメモリのままでCPUユニットの動作を継続させ、

プログラミングツールにて、ユーザプログラムの編集を終了した後、CPUユニットがサイクリック処理動作中に、前記一方のメモリへ編集後のユーザプログラムをダウンロードして格納し、

CPUユニットにて、その格納を終了した後、前記ユーザプログラムの演算実行対象を前記他方のメモリから前記一方のメモリに切り替えて、CPUユニットが新しく編集したユーザプログラムを実行処理するようにしたことを特徴とするプログラマブルコントローラのユーザプログラム編集方法。

【請求項 9】

ユーザプログラムを格納する 2つのメモリを持ち、I/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するプログラマブルコントローラにおいて、プログラミングツールによって前記ユーザプログラムをオンラインエディットされる際のプログラマブルコントローラの処理方法であって、

予め、前記 2つのメモリに格納しているユーザプログラムをそれぞれ同一の内容とし、ユーザプログラムの演算実行対象を 2つのメモリのいずれか一方のメモリに格納されているユーザプログラムと定めるとともに、プログラミングツールからの書き込み可能対象を他方のメモリのみと定め、

プログラマブルコントローラのサイクリック処理動作をそのまま継続させながら、

プログラマブルコントローラの周辺処理動作中に、前記プログラミングツールに対してどちらかのメモリに格納されているユーザプログラムを送信し、

プログラマブルコントローラの周辺処理動作中に、前記プログラミングツールからのユーザプログラムを前記他方のメモリへ格納し、

ユーザプログラムの演算実行対象を前記一方のメモリから前記他方のメモリに切り替えて、前記他方のメモリに格納されたユーザプログラムを実行処理するようにし、

プログラマブルコントローラの周辺処理動作中に、前記他方のメモリに格納されたユーザプログラムを前記一方のメモリへ格納して、2つのメモリのユーザプログラムを同一内容とするようにした

ことを特徴とするオンラインエディットされる際のプログラマブルコントローラの処理方法。

【書類名】明細書

【発明の名称】 制御装置、CPUユニット、プログラマブルコントローラのユーザプログラム編集方法、及びオンラインエディットされる際のプログラマブルコントローラの処理方法

【技術分野】**【0001】**

この発明は、ユーザメモリを演算実行するCPUユニットやプログラマブルコントローラにおけるユーザプログラム編集に関するものである。

【背景技術】**【0002】**

ファクトリーオートメーション（FA）の制御装置として、プログラマブルコントローラ（PLC）が用いられている。このPLCは、複数のユニットから構成される。すなわち、電源供給源の電源ユニット、PLC全体の制御を統率するCPUユニット、FAの生産装置や設備装置の適所に取り付けたスイッチやセンサの信号を入力する入力ユニット、アクチュエータなどに制御出力を出す出力ユニット、通信ネットワークに対する処理、例えばプログラミングツールや上位モニタ装置、他のPLC等に対してデータ通信する処理を行う通信ユニットなどの各種のユニットを適宜組み合わせて構成される。

【0003】

PLCのCPUユニットにおける制御は、まず初期処理をし、その後、入力ユニットで入力した信号をCPUユニットのI/Oメモリに取り込み（INリフレッシュ処理）、ユーザプログラム記述言語（例えばラダー言語）により組まれ、メモリに予め登録されたユーザプログラムに基づき論理演算をし（演算実行処理）、その演算実行結果をI/Oメモリに書き込んで出力ユニットに送り出し（OUTリフレッシュ処理）、その後、いわゆる周辺処理を行う。そして、INリフレッシュ処理と演算実行処理とOUTリフレッシュ処理と周辺処理とをサイクリックに繰り返し処理するようになる。なお初期処理の後のサイクリック処理の順は別の変形例でもよく、例えば、OUTリフレッシュ処理とINリフレッシュ処理（まとめてI/Oリフレッシュ処理とも言う）、演算実行処理、周辺処理でもよい。この場合、最初のOUTリフレッシュ処理は初期処理による固定値を出力し、次サイクルのOUTリフレッシュ処理から演算実行結果を出力することとなる。また、サイクリック処理の順を、演算実行処理、I/Oリフレッシュ処理、周辺処理とする変形例でもよい。この場合、最初の演算実行処理は初期処理による固定値の演算を行い、次サイクルの演算実行処理からINリフレッシュ処理で取り込んだINデータに基づく演算を行うこととなる。

【0004】

ユーザプログラムは、PLCに対してネットワーク接続されたプログラミングツールによって作成または編集される。作成後または編集後のユーザプログラムは、プログラミングツールからPLCへダウンロードされ、CPUユニットのユーザメモリに格納される。このユーザプログラムは、いわゆるラダー言語で構成されたラダープログラムによって複数の入力接点の論理回路が組まれている。なお、ユーザプログラムには、ひとつかたまりの動作をプログラミングしたファンクションブロックを構成要素に含んでいる場合もある。ファンクションブロックは、複数の演算回路を組み合わせてひとつに部品化されたものである。

【0005】

ところで、実際にPLCのCPUユニットにてユーザプログラムを実行（サイクリック処理の演算実行）し、生産設備や設備装置を稼動した後においても、プログラム上に不具合を発見したり、動作の不具合が発生したりすることがある。その場合には、ユーザプログラムに対して追加・削除・変更などの編集を行う必要が生じる。係るプログラムの編集処理の一態様として、PLCの制御を実行しつつ、プログラムの編集を行うオンラインエディットがある。

【0006】

従来、このオンラインエディットを行う場合には、プログラミングツールによってユーザプログラム全体の中の編集対象のプログラム部分の直前にジャンプ命令等を一時的に追加するようにしていた。そして、PLCがユーザプログラムを演算実行する際に、編集対象部分だけを飛ばしてユーザプログラムを実行することにより、PLCを停止することなく動作を継続できていた。そしてツールにて、PLCが動作中の状態で、ユーザプログラムの編集対象部分に対し追加・削除・変更などの編集作業を行うようにしていた。なお、上記の発明と類似する先行技術としては、特許文献1がある。

【0007】

【特許文献1】特開2001-142510号公報

【発明の開示】

【発明が解決しようとする課題】

【0008】

しかしながら、上記した従来のオンラインエディットは、1回の処理で1つの回路に対して行うことは比較的簡単にできても、ユーザプログラムの複数箇所に点在するラダー回路を一度に修正するのは困難であった。その理由は、ユーザプログラム中の複数の編集箇所すべてにジャンプ命令を一時的に追加するのは大変面倒であったからである。また編集後に、一時的に追加したジャンプ命令等をすべて削除する必要がある面倒であった。さらにジャンプ命令をひとつでも削除し忘れた場合には、編集したプログラムが正常に動作しなくなり、不具合が生じていた。

【0009】

また、ファンクションブロックに対するオンラインエディットも従来実行できなかった。すなわち、ファンクションブロックはユーザプログラム中の一箇所にまとまって格納されることはまれで、複数箇所に分散していることが多い。これは、ファンクションブロックの構成要素が、メインのラダープログラムに対してサブルーチンプログラムのようになっていることと、インスタンスとボディとからなっていることから、通常はそれらがプログラム中の複数箇所に分散して格納されるからである。よって、従来の通常の回路に対する編集の方法をファンクションブロックに適用すると、前述と同様に、ジャンプ命令を挿入する作業が多く発生してしまい、その作業が大変面倒であった。またジャンプ命令を削除するのも面倒で、そのジャンプ命令をひとつでも削除し忘れると編集したファンクションブロックプログラムが正常に動作しないこととなり、不具合であった。

【0010】

ファンクションブロックの事情をもう少し詳しく述べる。ユーザプログラムを格納するユーザプログラムメモリ全領域は、ファンクションブロックのボディ部が複数格納されるファンクションブロックレコード部（記録領域）と、実際のメインのラダープログラム部分とファンクションブロックが挿入されている場合にはそれに対応するファンクションブロックの入力・出力を特定するインスタンスとが格納されるプログラムレコード部（記憶領域）とに区分けされている（詳しくは図8でも述べる）。よってファンクションブロックに対して編集処理を行うということは、入力と出力のインスタンス部分と、実際のファンクションブロックのボディ部分との複数箇所を同時に編集することとなる。よって、上述のメインユーザプログラムのラダー回路の複数箇所を一度に修正する場合と同じく、ジャンプ命令挿入および削除作業が多く発生してしまうこととなる。

【0011】

オンラインエディットのもうひとつの不具合を言うと、ユーザプログラム中の編集対象部分はジャンプ命令によって処理がスキップされるので、制御に必要不可欠な動作に対応するプログラム部分に対して編集処理すると、スキップ処理に伴ってPLC動作、ひいては生産設備の動作に悪影響を与えるおそれがある。従って、プログラムのどの部分に対しても自由にオンラインエディットが行えるとは限らない。

【0012】

この発明は、ユーザプログラムの編集対象がどこであっても、また編集箇所が多くなっても（ファンクションブロック編集も含む）、オンラインエディットによる編集処理が行

えるCPUユニット及びユーザプログラム編集方法を提供することを目的とする。

【課題を解決するための手段】

【0013】

この発明による制御装置は、I/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するCPUユニット(10)を含むプログラマブルコントローラと、そのプログラマブルコントローラに接続されてユーザプログラムを編集するためのプログラミングツール(10)とからなり、プログラミングツールは、プログラマブルコントローラに対してユーザプログラムをアップロードおよびダウンロードする手段と、ユーザプログラムを編集する手段と、を持ち、プログラマブルコントローラのCPUユニットは、ユーザプログラム格納用の2つのメモリ(UM14, T-UM19)と、サイクリック処理動作中にプログラミングツールからユーザプログラム編集処理がなされる場合に、ユーザプログラム実行処理の実行対象をいずれか一方のメモリに格納されているユーザプログラムと定める手段(MPU12)と、プログラミングツールのアップロードによって、サイクリック処理動作の周辺処理期間中にプログラミングツールに対して一方または他方のメモリのいずれかに格納されているユーザプログラムを出力する手段(MPU12, 通信I/F18)と、プログラミングツールからダウンロードされてきたユーザプログラムをサイクリック処理動作の周辺処理期間中にユーザプログラム実行処理の実行対象としていない他方のメモリへ格納する手段(MPU12)と、その格納が完了したことを条件にユーザプログラム実行処理の実行対象を一方のメモリから前記他方のメモリに切り替える手段(MPU12)と、他方のメモリに格納されたユーザプログラムを一方のメモリへコピー格納する手段(MPU12)とを持つ構成とした。

【0014】

また、この発明によるCPUユニット(10)は、プログラミングツールとの間でユーザプログラムをアップロードおよびダウンロードでき、I/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するもので、ユーザプログラム格納用の2つのメモリ(UM14, T-UM19)と、サイクリック処理動作中にプログラミングツールからユーザプログラム編集処理がなされる場合にユーザプログラム実行処理の実行対象をいずれか一方のメモリに格納されているユーザプログラムと定める手段(MPU12)と、プログラミングツールのアップロードによりサイクリック処理動作の周辺処理期間中にプログラミングツールに対して一方または他方のメモリのいずれかに格納されているユーザプログラムを出力する手段(MPU12, 通信I/F18,)と、プログラミングツールからダウンロードされてきたユーザプログラムをサイクリック処理動作の周辺処理期間中にユーザプログラム実行処理の実行対象としていない他方のメモリへ格納する手段(MPU12)と、その格納が完了したことを条件にユーザプログラム実行処理の実行対象を一方のメモリから他方のメモリに切り替える手段(MPU12)と、他方のメモリに格納されたユーザプログラムを一方のメモリへコピー格納する手段(MPU12)と、を持つ構成とした。

【0015】

一方、本発明に係るユーザプログラム編集方法は、ユーザプログラムを格納する2つのメモリ(UM, T-UM)を持ってI/Oリフレッシュ処理とユーザプログラム実行処理と周辺処理とをサイクリックに処理するCPUユニット(10)が接続されたプログラマブルコントローラに対して、プログラミングツール(20)からユーザプログラムを編集する方法であって、CPUユニットにて、予め前記2つのメモリに格納しているユーザプログラムを同一の内容としておき、プログラミングツールからユーザプログラムを編集処理するのに先立ち、ユーザプログラムの演算実行対象をいずれか一方のメモリに格納されているユーザプログラムと定めておき、CPUユニットのサイクリック処理動作をそのまま継続させ(図7(a)または図10(a))、プログラミングツールにてCPUユニットの動作中に、CPUユニットの一方または他方のメモリのいずれかに格納されているユーザプログラムをアップロードし(図4のST11)、そのユーザプログラムを編集し(図4のST12)、その編集を終了した後(図4のST13)、他方のメモリへ編集後の

ユーザプログラムをダウンロードして格納し（図4のST14、図6のST23やST24、図7のb、図10のb）、CPUユニットにて、その格納を終了した後（図6のST25）、ユーザプログラムの演算実行対象を一方のメモリから他方のメモリに切り替えてCPUユニットが編集後のユーザプログラムを実行処理するようにし（図6のST26、図7c、図10c）、他方のメモリに格納された編集後のユーザプログラムを一方のメモリに格納して両メモリのユーザプログラムを同一内容とする（図6ST27、図7d、図10e）ようにした。

【0016】

また、新しく次回、プログラミングツールにてユーザプログラムを編集する際、CPUユニットにて、ユーザプログラムの演算実行対象を他方のメモリから一方のメモリに切り替え、その状態を編集後のユーザプログラムのダウンロードが終了するまで維持するようにしてもよい。これは実施形態の図7dの状態から図7a、図7bへの処理流れに対応する。

【0017】

また、新しく次回、プログラミングツールにてユーザプログラムを編集する際の変形例として、CPUユニットにて、ユーザプログラムの演算実行対象を他方のメモリのままでCPUユニットの動作を継続させ、プログラミングツールにて、ユーザプログラムの編集を終了した後、CPUユニットがサイクリック処理動作中に一方のメモリへ編集後のユーザプログラムをダウンロードして格納し、CPUユニットにて、その格納を終了した後、ユーザプログラムの演算実行対象を他方のメモリから一方のメモリに切り替えて、CPUユニットが新しく編集したユーザプログラムを実行処理するようにすることでもよい。これは実施形態の図10eの状態から図10f、それ以降の処理に該当する。

【0018】

また、本発明に係るユーザプログラムをオンラインエディットする際のプログラマブルコントローラの処理方法は、予め、CPUユニットの2つのメモリに格納しているユーザプログラムをそれぞれ同一の内容とし、ユーザプログラムの演算実行対象を2つのメモリのいずれか一方のメモリに格納されているユーザプログラムと定めるとともに、プログラミングツールからの書き込み可能対象を他方のメモリのみと定め（図7a、図10a）、CPUユニットのサイクリック処理動作をそのまま継続させながら、CPUユニットの周辺処理動作中に、プログラミングツールに対して2つのメモリのいずれかに格納されているユーザプログラムを送信し、プログラミングツールから編集後のユーザプログラムを他方のメモリへ格納し（図6のST23、24、図7b、図10b）、ユーザプログラムの演算実行対象を一方のメモリから他方のメモリに切り替えて編集後のユーザプログラムを実行処理するようにし（図6のST26、図7c、図10c）、CPUユニットの周辺処理動作中に他方のメモリに格納された編集後のユーザプログラムを一方のメモリへ格納して2つのメモリのユーザプログラムを同一内容とする（図6のST27、図7d、図10e）ようにした。

【0019】

CPUユニットのユーザメモリに格納されたユーザプログラムに対してオンラインエディット編集をする場合、2つのユーザメモリには同一内容のユーザプログラムが格納されているので、一方のメモリのユーザプログラムを編集処理をしている間、他方のユーザメモリのユーザプログラムを演算実行に使うことができる。これにより、CPUユニットが演算実行するときに参照するユーザプログラムそのものに対しては編集処理をしないので、何ら制限無く編集処理と演算実行処理の両方が行える。

【0020】

そして、オンラインエディット編集が完了したならば、今まで編集対象になっていたほうのメモリを演算実行対象に切替えることにより、編集された新しいユーザプログラムに基づく演算処理がすぐに行える。

【発明の効果】

【0021】

以上のように、この発明では、CPUユニットにユーザメモリに加えてテンポラリユーザメモリを設け、その2つのメモリに同一のユーザプログラムを格納しておき、プログラミングツールが一方のユーザメモリに対してオンラインエディット編集処理をしている際に、CPUユニットが他方のテンポラリユーザメモリを参照してユーザプログラム実行処理を行う。オンラインエディット対象のユーザプログラムの格納場所とCPUユニットにおけるユーザプログラム演算処理対象のユーザプログラムの格納場所とが独立しているので、自由にオンラインエディット処理ができるようになる。よって、従来のオンラインエディット処理時の不具合はなくなり、編集対象が制御に必須の部分であっても、また編集箇所が多くなっても（ファンクションブロック編集も含む）、オンラインエディットによる編集処理が行える。

【発明を実施するための最良の形態】

【0022】

図1は、システム全体を示している。CPUユニット10は、PLCの構成要素の一つである。そして、CPUユニット10の本来のサイクリック処理動作をするためのシステムプログラムは、システムROM11に格納されている。そして、MPU12において、ワークメモリとしてのシステムRAM13を適宜使用しながら、システムROM11のシステムプログラムに従った所定の処理を実行する。また、PLCの運転時において、ユーザがプログラミングツールを用いて予め作成したユーザプログラムは、ユーザメモリ14に格納されている。さらに、ユーザプログラム実行の論理演算に用いるI/Oデータやパラメータは、I/Oメモリ15に格納されている。そしてCPUユニット10は、ユーザプログラム実行処理の前のINリフレッシュ処理においてINデータを、PLCシステムバス17を介して接続されたI/Oユニット（図示なし）から取り込む。そしてCPUユニット10における論理演算処理ではそのINデータに基づいて論理演算し、OUTリフレッシュ処理にてその論理演算結果をOUTデータとしてI/Oユニットに対して出力する。

【0023】

CPUユニット10におけるユーザプログラム実行処理は、MPU12と命令実行エンジン16とで行われる。MPU12は、ユーザプログラム実行処理に入ると、命令実行エンジン（ASIC）16に対してユーザメモリ14に格納されたユーザプログラム（命令オブジェクトコード）を順次呼び出して実行するように指示をする。その指示によって命令実行エンジン16は、呼び出した命令オブジェクトを解析し、命令実行エンジン16がもつ命令テーブルを参照し、プログラム命令実行をする。命令テーブルには、命令実行エンジン16が自ら実行できる基本命令（ANDやORの論理演算、簡単な数値演算など）が定義されている。命令テーブルにない応用命令（複雑な計算制御など）については、命令実行エンジン16自らは実行できない。そこで、命令実行エンジン16は、MPU12に対して、応用命令の実行依頼と、その実行に必要なINデータやパラメータのデータ送信とを行う。このように命令実行エンジン16とMPU12とで命令実行の分担をするのは、基本命令を得意とするエンジンと応用命令を得意とするMPUとによって処理の最大効率を図るためである。変形例として、MPU12の機能と命令実行エンジン16の機能とをひとつのプロセッサで実現するような構成としてもよい。他の変形例として、命令実行エンジン16が応用命令の一部についても実行できるようにしてもよい。（以下の説明では、ユーザプログラムは、命令実行エンジン16が実行対処できるものであることとする）。

【0024】

ユーザプログラム実行時に、命令実行エンジン16は適宜I/Oメモリ15にアクセスし、INデータを読出したり、OUTデータを書き込んだり、またはパラメータを読出して取得する。通信インタフェース18は、ツール20との間でデータの送受を行うものである。この通信インタフェース18を介してツール20は、CPUユニット10のI/Oデータをモニタしたり、ユーザメモリ14に格納されているユーザプログラムをアップロードしたり、編集したユーザプログラムをユーザメモリ14に対してダウンロードしたりす

る。このツール 20 との間の通信処理は、CPU ユニット 10 におけるサイクリック処理の周辺処理にて行う。

【0025】

本実施の形態の PLC は、ファンクションブロック (FB) が実行できる。図 2 (a) はメインのユーザプログラムの一部であり、ファンクションブロックは図 2 (a) に示すようにラダープログラム中に挿入されている。このファンクションブロックは、引数とも呼ばれる入力パラメータ (図示の例では「D0, D1」) と、戻り値とも呼ばれる出力パラメータ (図示の例では「D2, H1.0」) と、実際に演算実行する処理を規定するボディ B とを備える。ボディ B の中身は、例えば図 2 (b) に示すようになっていて、ひとかたまりの動作をするためのラダー言語でかかれたプログラムからなっている。メインのユーザプログラム中では、図 2 (a) に示すようにファンクションブロックのボディ部は箱で記述される。このボディ部の中身は、図 2 (b) に示すような複数の回路からなるプログラムからなっている。実際のユーザプログラム演算実行に際してファンクションブロックのところまで来ると、ファンクションブロックのボディ部が呼び出され、入力パラメータに基づいて実行され、その実行結果が出力パラメータとして出力される。

【0026】

ユーザプログラムの実行フローを図 3 に示す。CPU ユニット 10 の命令実行エンジン 16 は、ユーザメモリ UM 14 のユーザプログラムを上から順番に実行していく (ST1)。ユーザプログラム中に図 2 (a) のようにファンクションブロックが含まれていて、そのファンクションブロックを実行する順番になれば、命令実行エンジン 16 は、ファンクションブロックの呼び出しをする (ST2)。これでユーザメモリのファンクションブロックレコード部の該当 FB 記憶領域にジャンプする。そして、命令実行エンジン 16 は、インスタンス前処理に移る。インスタンス前処理では、ファンクションブロックの入力パラメータの値を IO メモリ 15 から読み出して、IO メモリ 15 のファンクションブロック実行領域にその値をコピーする (ST3)。このようにコピーするのは、ファンクションブロック用の IN データおよび OUT データの記憶領域を別途設けているからである。次いで、FB ボディ処理として、命令実行エンジン 16 は、ファンクションブロック実行領域を参照してボディに記述されたラダープログラム (図 2 (b) 等) を実行し、その実行結果を IO メモリ 15 のファンクションブロック実行領域へ格納する (ST4)。その後、インスタンス後処理として、IO メモリ 15 のファンクションブロック実行領域に記憶されている実行結果の値を出力パラメータ領域にコピーし、ファンクションブロック部分の処理を終了する (ST5)。次いで、命令実行エンジン 16 は、ユーザメモリ UM 14 を再び参照し、ファンクションブロックに続くユーザプログラムのメインルーチンの続きを実行する (ST6)。なお、係るファンクションブロックの実行処理手順自体は、従来公知のものであるので、その詳細な説明を省略する。

【0027】

ここで、本発明では、通常ユーザメモリ (表ユーザメモリ) 14 に加え、テンポラリユーザメモリ (裏ユーザメモリ) 19 を設けた。ユーザメモリ 14 は、通常動作時のユーザプログラムを格納する領域であり、そして、命令実行エンジン 16 はこのユーザメモリ 14 からユーザプログラムを呼び出して実行する。また、プログラミングツール 20 からのアクセス許可領域は、ユーザメモリ 14 であり、もう一方のテンポラリユーザメモリ 19 に対してはアクセスできないようになっている。これは、ユーザプログラムを参照する場所とツール 20 がアクセスする場所とを区別できるようにするためである。詳しくは図 6, 7 などで後述するが、つまり、命令実行エンジン 16 がユーザプログラムを実行する際に、その実行中にツール 20 により編集された直後のプログラムを参照しないようにするためである。もし命令実行エンジン 16 が実行中に編集プログラムを随時参照すると、プログラム内容が実行途中で変更されることとなり、動作が不安定になるからである。

【0028】

また、テンポラリユーザメモリ 19 は、ユーザメモリ 14 に格納されたユーザプログラムと同一のプログラムが格納されている。オンラインエディット時には、命令実行エンジ

ン16はユーザメモリ14を参照して実行せずに、代わりにテンポラリユーザメモリ19に格納されたユーザプログラムを呼び出して実行する。この切り替えにより、オンラインエディット中も、オンラインエディット前のユーザプログラムと同一内容のユーザプログラムに基づいた演算処理を安定して続けることができる。

【0029】

なお、ユーザメモリ14とテンポラリユーザメモリ19は、物理的に異なるメモリ（記憶素子）を用いても良いし、同一のメモリを用いてメモリ割付によって2つのメモリ領域を設ける形態で実現しても良い。

【0030】

さて、オンラインエディット時のツール20の操作・動作を図4に沿って説明する。まず、編集対象のユーザプログラムをCPUユニット10のユーザメモリ14からアップロードをする（ST11）。このときCPUユニット10側はオンラインエディット中の処理をしている。詳しくは図6で説明するが、このときCPUユニット10が実行対象としているユーザメモリはテンポラリユーザメモリ19のほうなので、アップロードする対象メモリはもう一方のユーザメモリ14であるほうが望ましい。これは、CPUユニット10が参照しているメモリとプログラミングツール20が参照しているメモリとを区別したほうが制御にとって安全だからである。ユーザメモリ14のプログラムが格納されている全ての領域についてアップロードが完了すると、ツール20にてオペレータがユーザプログラムを編集する。この編集というのは、ユーザプログラムの追加、削除、変更などを行うことである（ST12）。ユーザプログラムの編集が終われば、ツール20からCPUユニット10のユーザメモリ14へ編集後のユーザプログラムをダウンロードする（ST14：図5（a）参照）。ツール20からのダウンロードを受けるCPUユニット10にとっては、このダウンロードはサイクリック処理中の周辺処理期間にて行われることとなる。ダウンロードするプログラムの容量が多い場合には、複数回サイクルの周辺処理にまたがって分割してダウンロードを行っても良い。

【0031】

ツール20は、ダウンロードが正常に完了するかどうかを確認する（ST15）。その確認は、PLC側でダウンロード完了かどうかを自己判断し、完了したならばツール20に対してその旨を返信し、その返信をツール20が受信することでもよい。PLC側の自己診断は、例えばMPU12がユーザメモリ14に格納されたコードデータのサム値チェックなどにより正常にダウンロード完了できたか否かを判断する。もし正常に完了できていない場合には、ツール20が、MPU12からダウンロード不完全である旨の通知を受け、再度ダウンロードを実行するようになっている。

【0032】

この後、ユーザメモリ14にダウンロードしたユーザプログラムをテンポラリユーザメモリ19にコピーして記憶保持する。PLC側でダウンロードが完了したか否かを自己診断する場合には、このユーザプログラムコピー格納処理は、MPU12がダウンロードの正常完了を判断することによって開始される。この場合のコピー格納処理は、CPUユニット10におけるサイクリック処理中のユーザプログラム実行期間以外、例えば周辺処理期間で行われる。また変形例として、ツール20が編集プログラムをCPUユニット10へダウンロードする際に、ツール自身がMPU12に対して、ユーザメモリ14への格納処理命令とテンポラリユーザメモリ19へのコピー格納処理命令とを発信するようにしてもよい。（図5（b）参照）

【0033】

次に、図6に示すフローチャートに沿って、CPUユニット10側のオンラインエディット中の処理について説明する。ここではファンクションブロックの追加を、CPUユニットのサイクリック処理動作中に実行する場合を例とする（具体的な編集内容は図9を説明する際に後述する）。まず、ツール20がオンラインエディット処理をするべく、ユーザプログラムのアップロードを行おうとする前に、CPUユニット10の命令実行エンジン16におけるプログラム実行対象をテンポラリユーザメモリ19に切り替える（ST2

1)。この切り替えは、命令実行エンジン16の実行対象をユーザメモリ14にするかテンポラリメモリ19にするかを示す特定フラグを書き換えることにより、実現できる。また変形例として、プログラミングツール20によってオンラインエディットする際に、プログラミングツール20から自動的に、CPUユニット10へ命令実行エンジン16の実行対象をテンポラリユーザメモリ19に切り替える命令を出力するようにしてもよい。いずれかの手法によってST21の切り替えを行うことにより、命令実行エンジン16に対しては、テンポラリユーザメモリ19に格納されたユーザプログラム（命令オブジェクト）が提供され、実際の制御が継続される。ユーザメモリ14の内容とテンポラリユーザメモリ19の内容が一致しているため、命令実行エンジン16は今まで処理していたのと同じ内容のユーザプログラムを継続して実行することができる。

【0034】

次に、プログラミングツール20によりユーザメモリ14に格納されている編集前のユーザプログラムをすべてアップロードされる（なお変形例として、編集対象となる部分だけをアップロードすることもある）。このアップロード処理はCPUユニットにおけるサイクリック処理中の周辺処理期間に行われる。ユーザプログラムの容量が多い場合には、複数サイクルの周辺処理期間に分割してツール20に対して送信してもよい。ツール20側では、このアップロード送信が終了してから実際のユーザプログラム編集処理に入る（前述の図4のST12）。次に、MPU12は、ユーザメモリ14の領域のうち、追加するファンクションブロックのボディプログラム領域を確保する（ST22：図7（a）参照）。ここでユーザメモリ14のメモリマップについて図8を用いて説明する。メモリマップは、図8に示すように、先頭にプログラムの基本情報（サム値など）を記録する領域があり、次に、使用されているファンクションブロックのボディ部のプログラム情報を記憶するFBレコード部と、ユーザプログラムおよび付随するファンクションブロックのインスタンスを格納するプログラムレコード部がある。ST22では、FBレコード部における未設定の領域を、今回追加するファンクションブロックの記憶領域として特定する。

【0035】

図9に示すように、所定位置にFB Z__4を追加する編集をユーザプログラムに対して行う際には、MPU12が、新規に追加するFB Z__4のためにFBレコード部の書き込む領域を特定して確保する。図8のようにFBレコード部は、予めファンクションブロックが数多く書き込めるように区分けされていて、その区分け領域のアドレスが小さい領域から順にファンクションブロックを格納してある。よって、新規なFB Z__4のための領域確保をすることは、まだFBが格納されていない領域のうちアドレス値が小さい領域を確保することとなる。このようにFB記憶領域を特定した後、ツール20側にてその確保領域に格納しようとするファンクションブロックを編集する。同時に、ユーザプログラム中における追加対象のファンクションブロックを実行する該当位置（プログラムレコード部のプログラムタスクの該当記憶領域の位置）に、その追加ファンクションブロックのために確保したFBレコード部の該当記憶領域の先頭アドレスを呼び出すプログラムも記述する。この呼び出しプログラムについては、ユーザが都度に入力してもよいし、ツール機能によって自動化してもよい。この場合、FB記憶領域を確保したときにツール20はその先頭アドレスを把握しているので、その情報を用いて呼出プログラムを自動作成することができる。

【0036】

プログラミングツール20において、ユーザプログラムの編集（ファンクションブロックの追加、インスタンスの設定を含む）が終わると、ユーザメモリ14の確保したFB記憶領域に対して、追加したFBボディブロックがダウンロードされてくる。MPU12は、受信したFBボディブロックをユーザメモリ14の確保しておいた領域に格納する。次に、ユーザメモリ14のプログラムレコード部に対して、ユーザプログラム中に追加ファンクションブロック記憶領域（今回、特定・確保したFB記憶領域）を呼び出すための呼出プログラムが、ツール20からダウンロードされてくる。MPU12は同じように確保

した領域に格納する（ST23、ST24：図7（b）参照）。なお、ここでは編集部分だけをダウンロードされて格納する処理を説明したが、変形例として、ツール20から編集したユーザプログラムを全部（編集箇所、編集していない箇所とも全部）についてユーザメモリ14にダウンロードしてもよい。なお、ツール20から編集プログラムをユーザメモリ14へダウンロードする処理は、CPUユニット10にとっては、MPU12が通信I/F18を介してなされる処理であり、サイクリック処理中の周辺処理期間に行われる処理となる。

【0037】

次に、MPU12にて、ユーザメモリ14のプログラム基本情報を参照し、サム値チェックなどにより正常にダウンロードしたか否かを判断する（ST25）。変形例として、この処理をMPU12に代わって、ツール20が行っても良い。その場合は、ツール20がユーザメモリ14のプログラム基本情報を読み出して、サム値チェックの処理をすることとなる。

【0038】

MPU12が、ダウンロード処理を正常に完了したと判断すると、ユーザメモリ14に記憶されたユーザプログラムが、図9（a）の状態から図9（b）に示すように変更されたことになる。そこで、MPU12は、命令実行エンジン16におけるプログラム実行対象をテンポラリユーザメモリ19からユーザメモリ14へ戻す（ST26：図7（c）参照）。この切り替えに伴い、命令実行エンジン16は、ユーザメモリ14に格納された編集後のユーザプログラムを実行することになる。なおこの切り替え処理は、CPUユニット10側で正常にダウンロード完了判断したことによって行われるが、ツール20側でダウンロード完了判断をするようにした場合にはツール20から切り替え命令をCPUユニット10が受信し、またはツール20から前述の切り替えフラグが更新されることによって行うようにしてもよい。またこの切り替え処理は、CPUユニット10におけるサイクリック処理のユーザプログラム実行期間以外の期間で行うこととなる。例えば周辺処理期間に行う。

【0039】

次いで、MPU12が、ユーザメモリ14の内容をテンポラリユーザメモリ19にコピーする処理を行う。（ST27：図7（d）参照）。実際には、一度にユーザメモリの全体をコピーすることはできないので、PLCのサイクリックな処理中の周辺処理時に少しずつコピーすることになる。このコピー処理に伴い、編集後のユーザプログラムがテンポラリユーザメモリ19にも記憶されることとなる。よって、次のオンラインエディット時にも、命令実行エンジン16の実行対象をユーザメモリ14からテンポラリユーザメモリ19に切り替えることで、ツール20がアクセスするユーザメモリ14とは区別できるので、命令実行エンジン16はその時点のユーザプログラムを安定してそのまま実行継続できる。

【0040】

なお、命令実行エンジンがより広範囲の処理に対応できるものであれば、上記のダウンロードされてきたユーザプログラムを受信してユーザメモリ14へ格納する処理や、ユーザメモリ14とテンポラリユーザメモリ19との参照切替え処理や、編集後のユーザプログラムをユーザメモリ14からテンポラリメモリ19へのコピー処理などを、MPU12からの指示を受けることによって命令実行エンジン16が行うようにしてもよい。また、MPU12の機能と命令実行エンジン16の機能をひとつのプロセッサで実現するようにした場合には、これらの処理はそのプロセッサが行うこととなる。

【0041】

また、上記した実施の形態では、編集処理の一態様であるファンクションブロックの追加について説明したが、ファンクションブロックの削除編集も同様に行える。もちろん、ファンクションブロック以外に、メインのユーザプログラムの通常のラダー演算部分の複数箇所を編集する場合にも同様に行える。そして編集対象となるプログラム数が1個でも適用はできる。つまり、いずれの場合にも編集処理をするに先立ち、プログラム実行をテ

ンポラリユーザメモリ19側に移す。これにより、ユーザメモリ14に対してオンラインエディットによる編集処理（アップロード、ダウンロード処理）をしても、命令実行エンジン16は、ツール20によってアクセスされるメモリ領域とは別のメモリ領域を参照するので、ユーザプログラム実行自体には影響を与えなくなる。

【0042】

なお、ユーザメモリとテンポラリユーザメモリの扱いについて、上記の実施形態ではオンラインエディット処理をするときに命令実行エンジンの実行対象をUMからT-UMに切り替えていたが、別の実現方法があるので変形例について説明しておく。その変形例では、オンラインエディット処理をはじめるときに、上記の実施形態（図6のST21）のように命令実行エンジンの参照対象のメモリを切り替えるのではなく、予めメモリの扱いの定義を変えておくことで実現できるようにした。

【0043】

オンラインエディット処理をするとき、命令実行エンジンが参照しているメモリは参照用メモリ（テンポラリメモリT-UM）と定義されていて、同一のユーザプログラムが格納されているもう一方のメモリ（今時点で命令実行エンジンが参照していないほうのメモリ）をエディット用メモリ（ユーザメモリUM）と定義されている（図10（a）参照）。そしてユーザプログラムを編集した後、エディット用メモリUMにその編集プログラムをダウンロードして格納する（図10（b）参照）。ダウンロードが完了した後、命令実行エンジンが参照するメモリを参照用メモリT-UMからエディット用メモリUMに切り替える。この切り替えによって、命令実行エンジンは編集後プログラムを参照して実行ようになる。またこの時点ではT-UMのほうのメモリには編集前のプログラムが格納されている（図10（c）参照）。ここまでは図7の説明と同じである。

【0044】

次に、各メモリの扱いの定義を変える。つまり、現時点でエディット用メモリUMであるほうのメモリを参照用メモリT-UMへ定義変更するとともに、参照用メモリT-UMであるほうのメモリをエディット用メモリUMへ定義変更する（図10（d）参照）。この定義変更によって、ツールは定義変更後の参照用メモリT-UMにアクセスできなくなる。そして、定義変更後のエディット用メモリUMには編集後のプログラムがまだ格納されていないので、参照用メモリT-UMに格納されている編集後プログラムをエディット用メモリUMにコピーする（図10（e）参照）。

【0045】

次回、オンラインエディット処理をするときには、ツールは、定義変更後のエディット用メモリUMに対して、エディット処理を行う。つまり、エディット用メモリUMに対し、そのFBボディ領域を確保することや、編集後のプログラムをダウンロードする（図10（f）参照）。ダウンロード完了すれば、前述と同様に命令実行エンジンの参照対象を切り替え、メモリの定義を入れ替え、編集後のプログラムをコピー格納する。このようにすることにより、ツール20がアクセスするメモリ（ユーザメモリUM）と、命令実行エンジン16が参照するメモリ（テンポラリユーザメモリT-UM）とは区別できるので、命令実行エンジン16はユーザプログラムを安定してそのまま実行継続できる。

【0046】

なお、プログラミングツール20からアクセスできるメモリをユーザメモリ14だけに制限した例を説明したが、ツールがメモリにアクセスする場合の変形例が考えられる。プログラミングツール20がユーザプログラムをアップロードする際には、ユーザメモリとテンポラリユーザメモリとは同一内容のユーザプログラムが格納されているので、命令実行エンジン16が参照対象としているほうのメモリでもよい。これはアップロードつまり読出し処理をすることによってメモリの内容が変化するわけではないからである。結局、アップロード時には、プログラミングツール20から、命令実行エンジン16が参照対象としているほうのメモリへも、参照対象でないほうのメモリへもアクセスしてもよい。いっぽう、編集後のユーザプログラムをダウンロードする際には、メモリ内容を書き換えることとなるので、命令実行エンジン16が参照対象していないほうのメモリに対してダウ

ンロードする必要がある。

【0047】

プログラマブルコントローラの変形例について説明しておく。上述の実施形態では、複数のユニットから構成される PLC を示した。この PLC は電源ユニット、CPU ユニット、入力ユニット、出力ユニット、通信ユニットなどの各種のユニットを組み合わせで構成されるものであった。この発明はそのような PLC に限らず、例えば、各ユニットの機能を 1 つのユニットにより一体化した PLC であってもよい。これはマイクロ PLC と呼ばれる 1 ユニット型プログラマブルコントローラである。このマイクロ PLC は通信ポートを備えているので、ネットワークを介してプログラミングツールを接続することができる。このようなプログラマブルコントローラであってもこの発明を適用することができる。

【図面の簡単な説明】

【0048】

【図 1】 本発明に係る CPU ユニットの一実施の形態を示すブロック図である。

【図 2】 ファンクションブロックを説明する図である。

【図 3】 ファンクションブロックを含むラダープログラムの実行処理を示すフローチャートである。

【図 4】 MPU の機能（ダウンロード）を示すフローチャートである。

【図 5】 MPU のダウンロードの作用を説明する図である。

【図 6】 MPU の機能（ファンクションブロックの追加）を示すフローチャートである。

【図 7】 MPU の機能（ファンクションブロックの追加）の作用を説明する図である。

【図 8】 ユーザメモリのメモリマップの一例を示す図である。

【図 9】 MPU の機能（ファンクションブロックの追加）の作用を説明する図である。

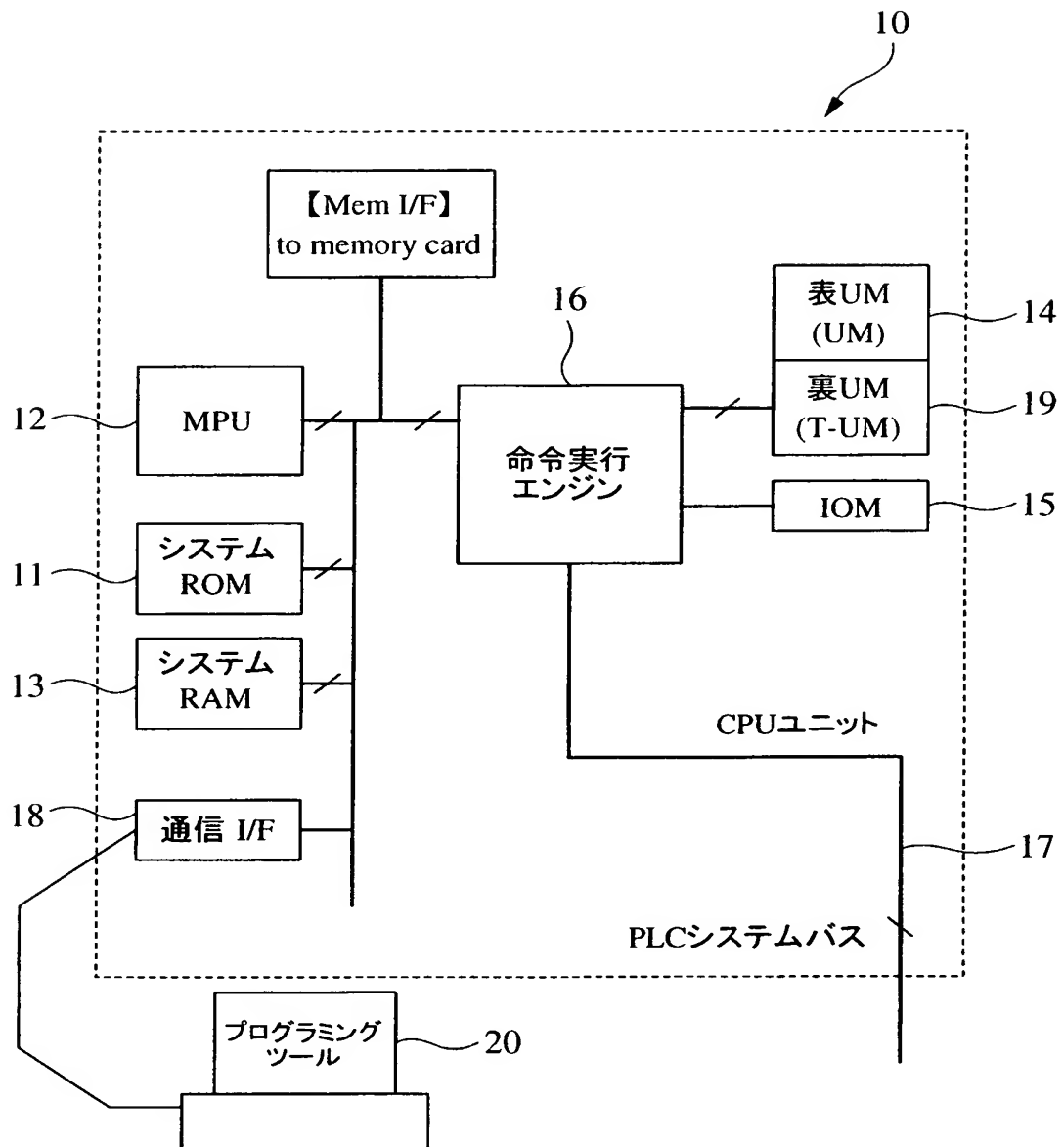
【図 10】 MPU の機能（ファンクションブロックの追加）の作用における変形例を説明する図である。

【符号の説明】

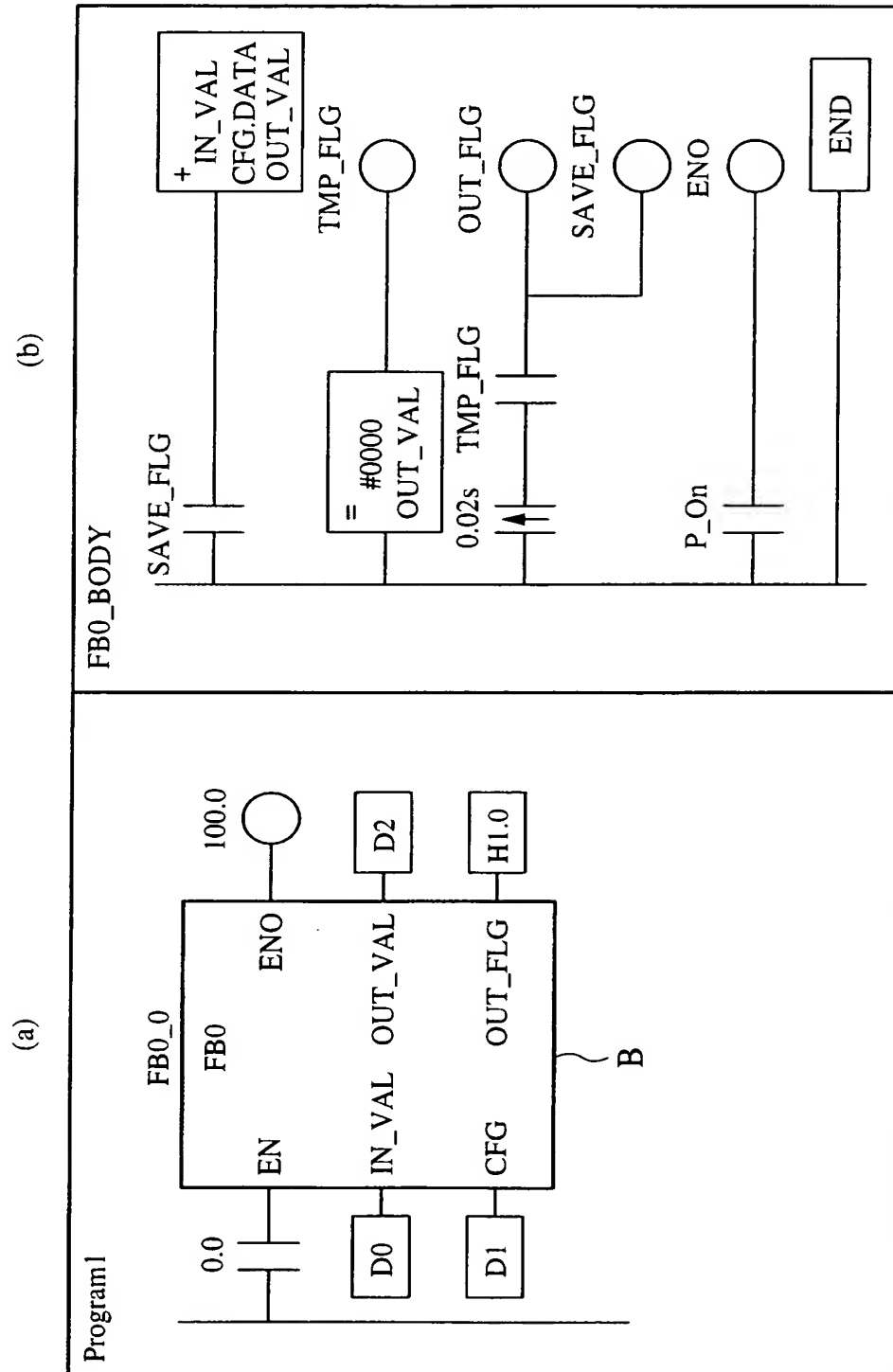
【0049】

- 10 CPU ユニット
- 11 システム ROM
- 12 MPU
- 13 システム RAM
- 14 ユーザメモリ
- 15 I/O メモリ
- 16 命令実行エンジン
- 17 システムバス
- 18 通信インタフェース
- 19 テンポラリユーザメモリ
- 20 ツール

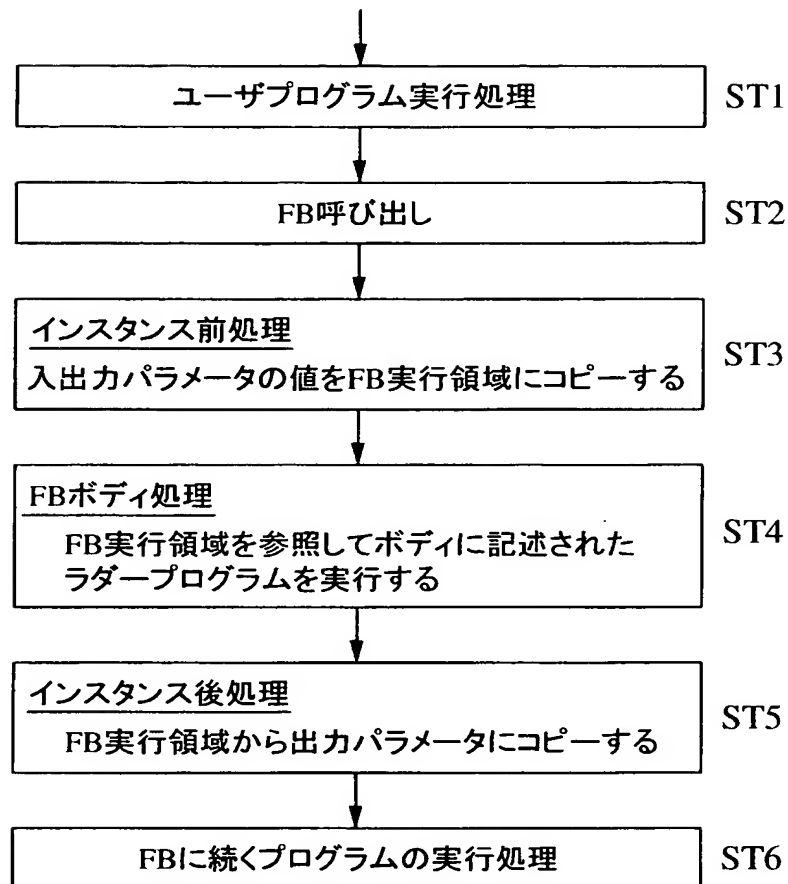
【書類名】 図面
【図 1】



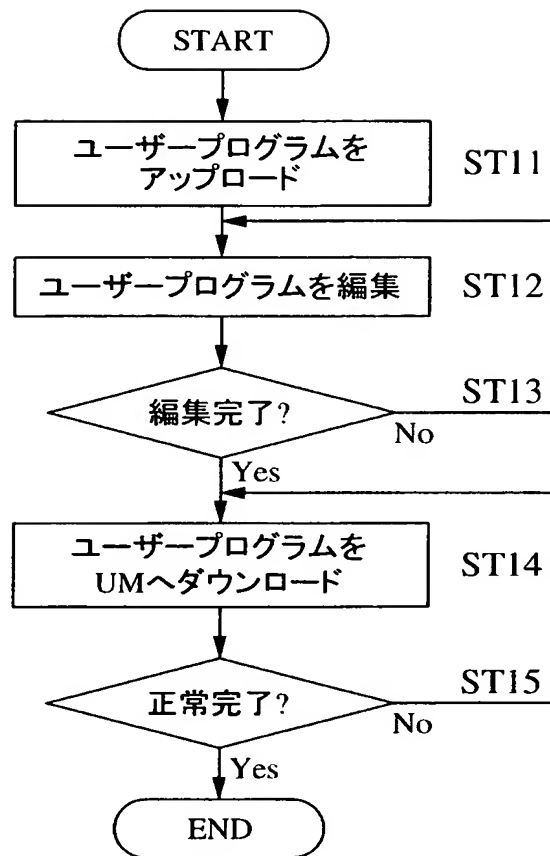
【図 2】



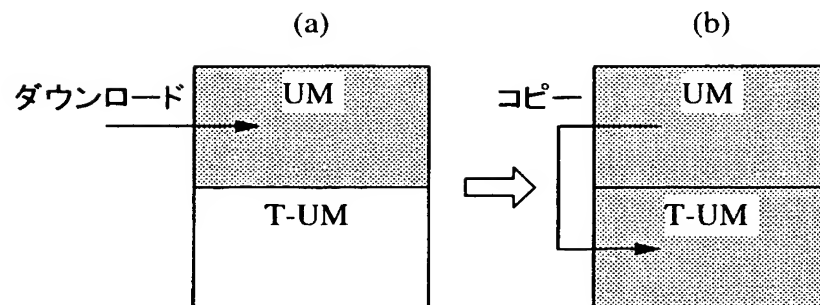
【図 3】



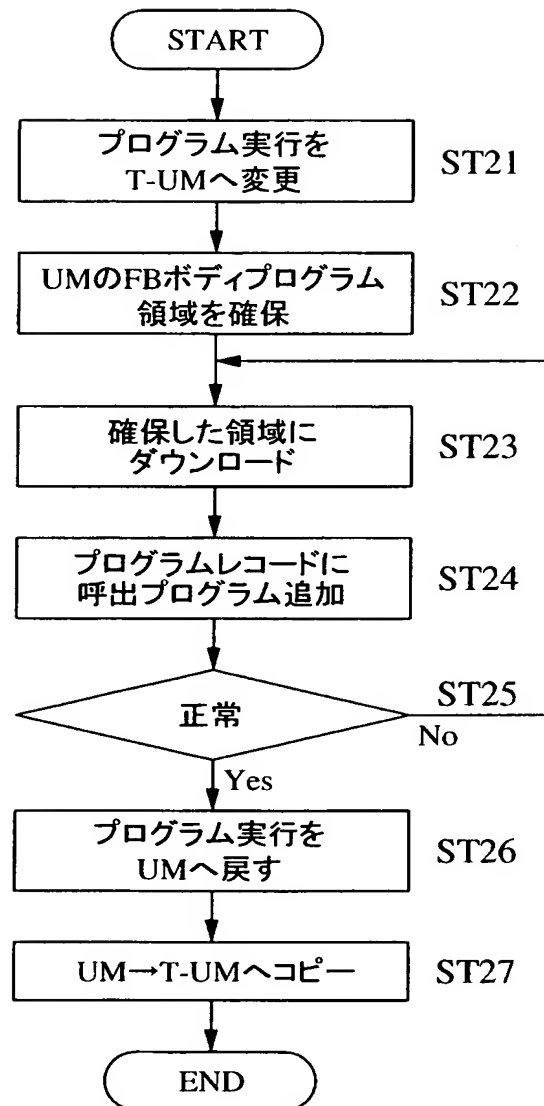
【図 4】



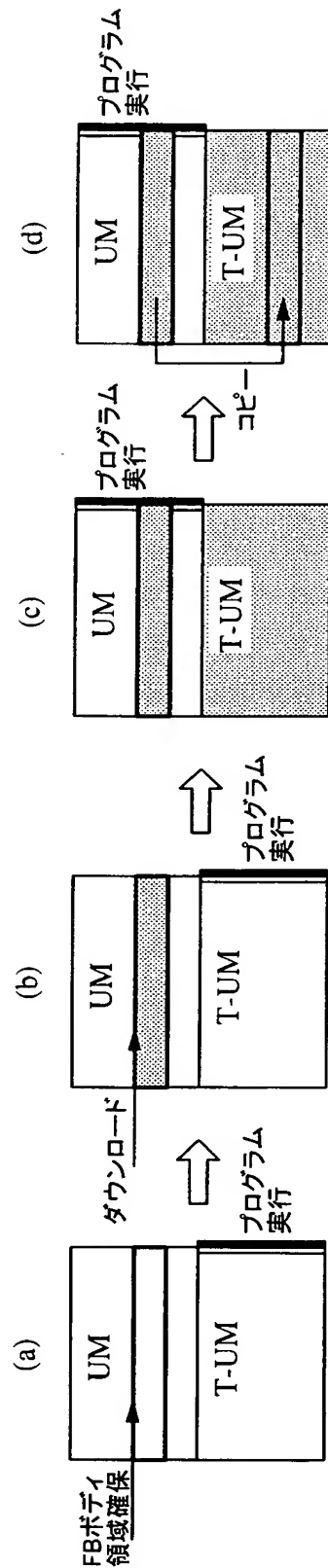
【図 5】



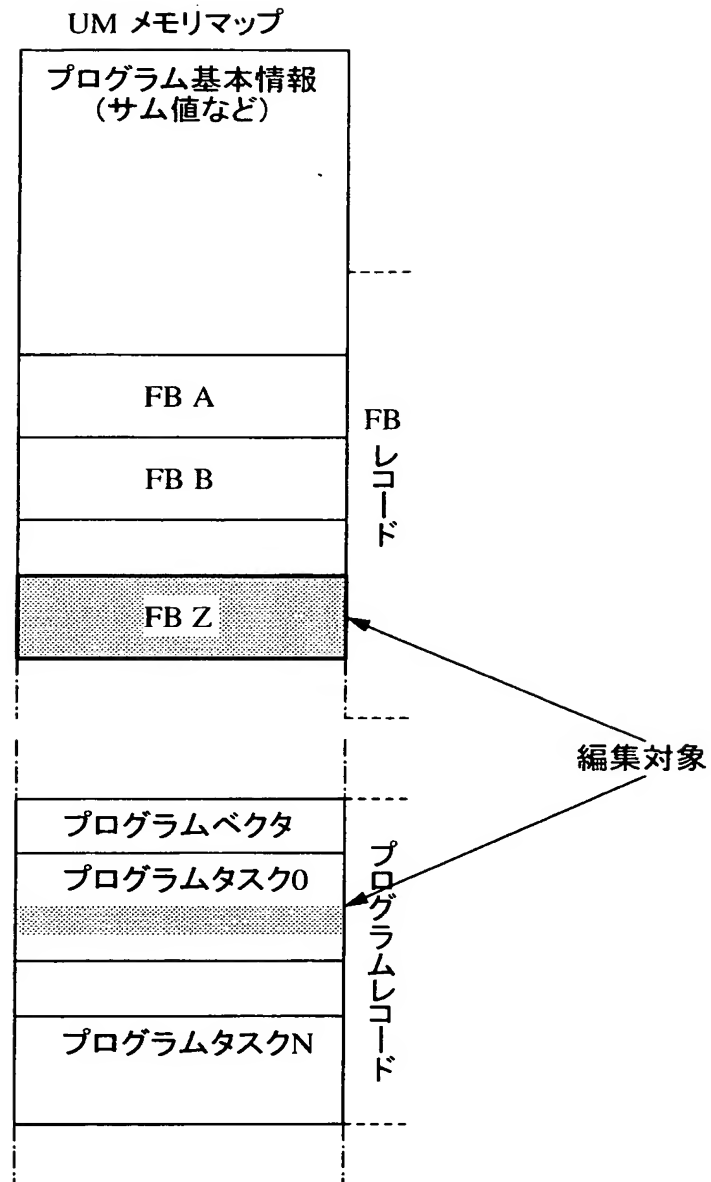
【図 6】



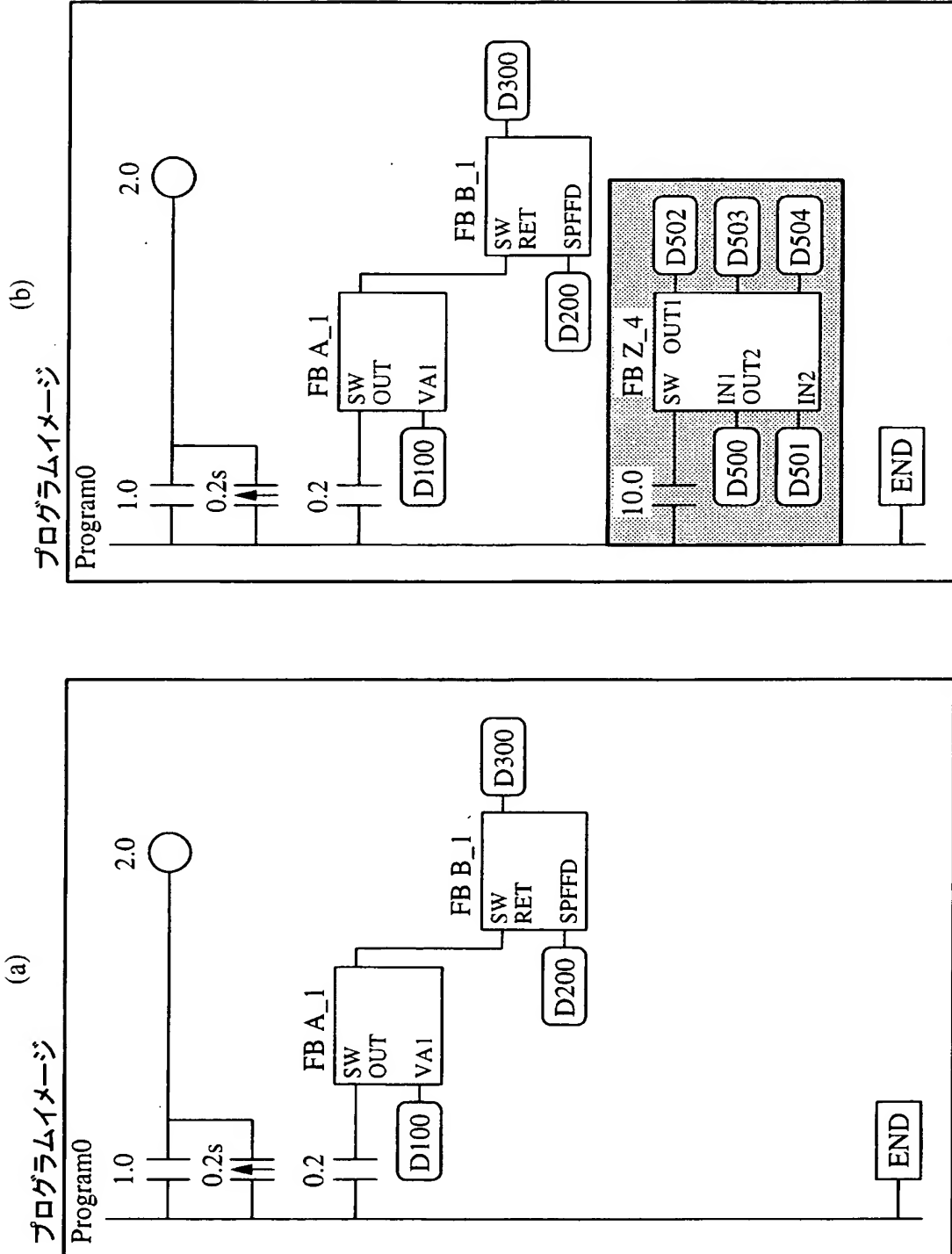
【図 7】



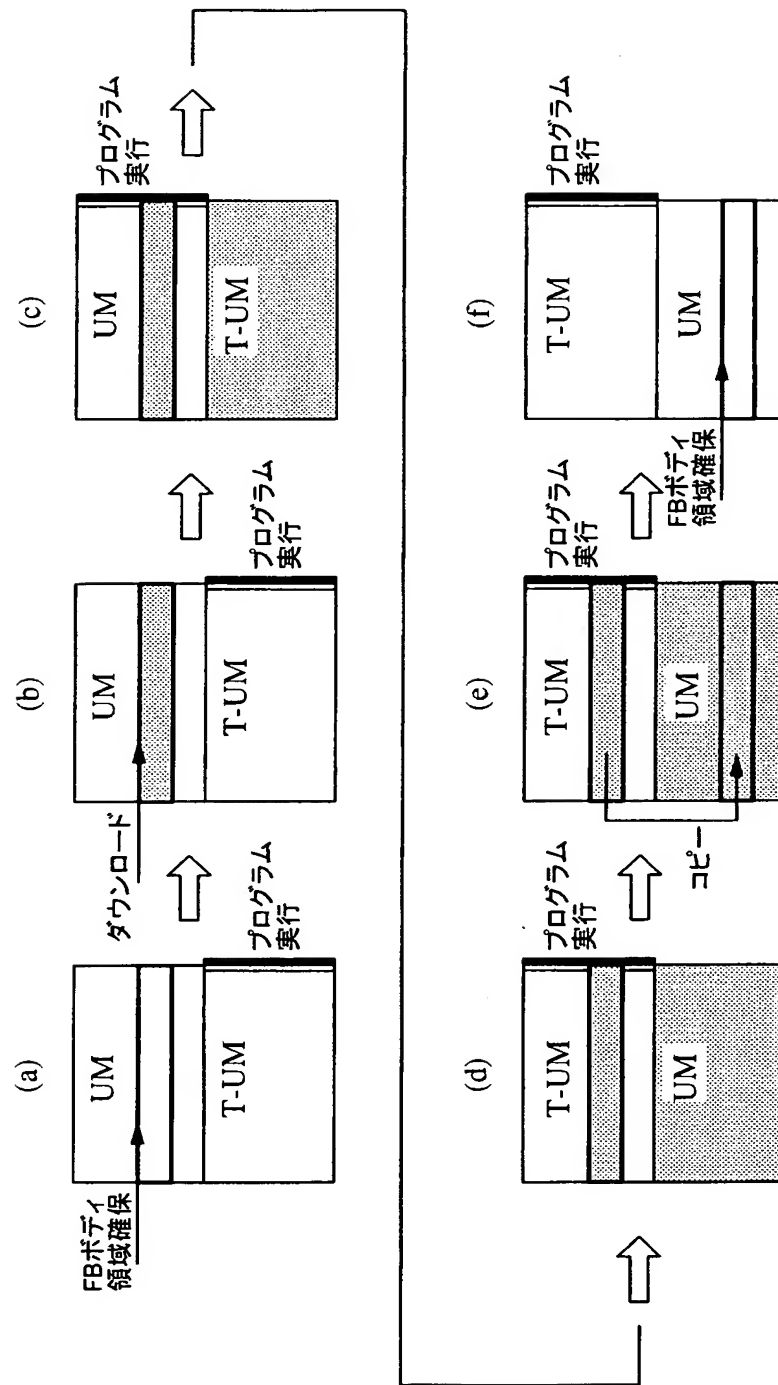
【図 8】



【図 9】



【図 10】



【書類名】 要約書**【要約】**

【課題】 オンラインエディットによるプログラムの修正が容易に行うことができ、かつ、編集対象（ファンクションブロックか否か）並びに個数に関係なく編集処理が行える CPU ユニットを提供すること

【解決手段】 ユーザメモリ 14 に格納されたユーザプログラムを演算実行するプログラマブルコントローラにおける CPU ユニット 10 である。そして、ユーザプログラムと同一のユーザプログラムを格納するテンポラリユーザメモリ 19 を設け、ユーザメモリに対する編集処理を行う際には、テンポラリユーザメモリに格納されたユーザプログラムが命令実行エンジン 16 に呼び出され、演算実行される。編集後は、ユーザメモリに格納された編集済みユーザプログラムに基づき実行され、その編集済みユーザプログラムは、テンポラリユーザメモリにコピーされる。

【選択図】 図 1

特願 2 0 0 4 - 0 5 9 8 6 5

出 願 人 履 歴 情 報

識別番号 [0 0 0 0 0 2 9 4 5]

1. 変更年月日 2 0 0 0 年 8 月 1 1 日

[変更理由] 住所変更

住 所 京都市下京区塩小路通堀川東入南不動堂町 8 0 1 番地
氏 名 オムロン株式会社